The Distributed Virtual Windtunnel

Steve Bryson[†] and Michael Gerald-Yamasaki[††]

RNR Technical Report RNR-92-010, March 1992

Applied Research Branch, Numerical Aerodynamics Simulation Division
NASA Ames Research Center
MS T27-A
Moffett Field, Ca. 94035
bryson@nas.nasa.gov

Abstract

An implementation of a distributed virtual environment for the shared interactive visualization of large unsteady three-dimensional flowfields is described. Computation of the visualizations are performed on a Convex C3240 computer, and the visualization data is transferred over a high-speed network to a Silicon Graphics Iris workstation for rendering. A boom-mounted six degree of freedom head-position-sensitive stereo CRT system is used for display. A hand position sensitive glove controller is used for controlling various tracers (e.g. "smoke") for the visualization of the flow. User commands are sent to the Convex, which interprets these commands and computes the corresponding visualization. With this architecture, several users may share and cooperatively control the visualization generated by the Convex. The distributed architecture is also interesting to those using conventional screen and mouse interfaces. This work extends the work of Bryson and Creon Levit [1] in the development of a virtual environment for the visualization of fluid flow to large data sets.

[†] Employee of Computer Sciences Corporation. Work supported under government contract NAS 2-12961
[††] Employee of NASA

1: Introduction

In a previous work [1], a virtual environment for the visualization of three-dimensional unsteady fluid flows was described. This system uses virtual environment interaction techniques, that allow fully three-dimensional display and control of computer generated graphics, to explore the global structure of pre-computed unsteady simulated flow fields. All control, computation and rendering in this previous work took place on a multi-processor high performance graphics workstation. In this environment single grid unsteady data sets were studied. Due to performance requirements, these data sets were limited to fewer than 240 megabytes in size. Interesting unsteady data sets are, however, considerably larger than this, often in the tens of gigabytes range. Visualization of these larger data sets using virtual environment techniques is not practicable on current high-performance workstations. This paper describes a distributed architecture for a virtual environment system designed to visualize these larger data sets.

The distributed virtual windtunnel will be discussed in the context of the unsteady flow around a tapered cylinder [2]. This data exhibits interesting vortical and recirculation phenomena. Each timestep consists of about on and a half megabytes of velocity data, and 800 timsteps were computed.

In the remainder of this section, the problem of visualizing unsteady flowfields is described. Section 2 describes the visualization techniques used in the stand-alone virtual windtunnel [1], the virtual environment system for flow visualization implemented on a stand-alone high-performance graphics workstation. Section 3 describes the virtual environment hardware used for control and display. Section 4 describes the distributed library [3] which is used for the communications between the graphics workstation and the host supercomputer. Section 5 describes the design of the distributed virtual windtunnel. Section 6 describes current performance. Further work and conclusions are described in section 7.

1.1 Unsteady Fluid Flow Visualization

Unless otherwise noted, a *flowfield*, is a numerical solution to a three-dimensional computational fluid dynamics (CFD) simulation, and in particular, the time-dependent velocity vector field part of that solution. Complicated geometrical and topological situations abound in such fields. Methods of visualizing these phenomena inspired by the visualization of real flows in real wind (or water) tunnels [5] can be used for simulated flow. Examples include smoke injection, dye advection, and time exposure photographs.

The flowfields considered in this paper are pre-computed solutions of the time-accurate Navier-Stokes equations of fluid motion. These unsteady flowfields are represented as a sequence of successive three-dimensional velocity vector fields. Each of these velocity vector fields is considered as a timestep.

1.2 Virtual Environments

Virtual environments [5] are a new approach to user interfaces in computer software. This approach involves integrating a variety of input and display devices to give the user the illusion of being immersed in an interactive computer generated environment. The computer generated scene is displayed in stereo to create the illusion of depth, and is rendered from a point of view that tracks the user’s head. The user also has an input device, typically an instrumented glove, which enables direct manipulation of objects in the computer generated environment.

Virtual environments provide a useful interface for analyzing unsteady fluid flow phenomena which involve complex three-dimensional structure. By providing the user with the illusion that the elements of the computer generated environment are real, interactive objects, the user becomes more directly involved in the investigation of the phenomena in that environment. By providing true three-dimensional control over objects in the environment, they can be directly manipulated to perform as the user desires. By providing head-tracked stereoscopic wide field of view displays, the three-dimensional structure of virtual objects can be unambiguously perceived.

The task of providing an illusion of reality places severe demands upon the entire computer system used to generate the virtual environment. Informal studies have shown that to sustain the illusion, the system must repeatedly react to the user's commands and display the virtual scene in stereo to the user in less than 1/8th of a second. Slower performance destroys the illusion, removing essentially all of the advantages of virtual environments. Thus the input of the user commands including user head position, the access to the data that is being visualized, the computation of the visualizations on that data, and the rendering of those visualizations from the user's point of view must all occur in less than 1/8th of a second. Faster performance is highly desirable, though a tradeoff must be made between a rich environment and frame rate. Ten frames/second will be taken as the desired frame rate.

These performance demands place strong constraints on the design of a system. The flow visualization techniques that can be used in a virtual environment are limited to those that can be computed in the time allowed. For example, interactive streamlines of a flow computed with fast integration methods can be used, but interactive isosurfaces, which require computationally intensive algorithms such as marching cubes, can not. The computer platform used in the virtual environment system must be sufficiently powerful and respond with sufficient speed. In a distributed implementation, there must be enough network bandwidth to deliver the data moving in both directions in the time required. The satisfaction of these constraints is the primary consideration in the design of the system described in this paper. Both streamline calculation and flowfield data I/O must be accomplished in less than 1/8 of a second to sustain sufficient frame rates for the virtual environment.

2: The Virtual Windtunnel

The stand-alone virtual windtunnel [1] has demonstrated that virtual environment techniques are useful in visualizing complex fluid flows. The stereo head-tracked display is a very effective way of displaying the complex three-dimensional features of a fluid flow. Input via an instrumented glove is a useful and intuitive way to position the various flow visualization tools. The idea is to create the illusion that the user is actually in the flow manipulating the visualization tools (figure 1). Unlike someone in a real flow field, however, the user's presence in no way disturbs the flow. Thus, sensitive areas of flow, such as boundary layers and chaotic regions, can be investigated easily. The flow can be investigated at any length scale, and with control over time. The time evolution of the flow can be sped up, slowed down, run backwards, or stopped completely for detailed examination.

The stand-alone virtual windtunnel is based on a Silicon Graphics Iris 380GT VGX system. This is a multiprocessor system with eight 33 MHz MIPS R3000 processors with R3010 floating point chips. The performance of the machine is rated at approximately 200 MIPS and 37 megaflops. Our system has 256 MBytes of memory. The VGX has a drawing speed rating of about 800,000 triangles/second.

The display device for this system is a boom-mounted six degree of freedom head-position-sensitive stereo CRT system. The control device is an instrumented glove which provides the position and orientation of the user's hand as well as the degree of bend of the user's fingers. These devices are the same as those used in the distributed virtual windtunnel, and are described in more detail in section 3.

2.1 Visualization Tools

As described in section 1, the tools considered in this paper for visualizing unsteady velocity vector fields are inspired by classical techniques used in real wind and water tunnels. Currently, the visualization methods, or tools, that we have implemented are streaklines, particle paths, and streamlines.

A *streakline* is formally defined as the locus of infinitesimal fluid elements that have previously passed through a given fixed point in space [4]. Streaklines are analogous to smoke or collections of bubbles (figure 1). A *particle path* is formally defined as the locus of points occupied over time by a given single, infinitesimal fluid element [4]. This corresponds to a “time exposure photograph” of the motion of a single small particle injected into the flow. A *streamline* is formally defined as the integral curve of the instantaneous velocity vector field that passes through a given point in space at a given time [4]. They provide direct insight into the instantaneous geometry of the flowfield (figures 2 and 3).

All of these techniques involve injecting virtual particles into the flow. We call the point of injection for a particular tool the *seed point* for that tool. Each of the techniques described above are computed by selecting a set of initial positions and integrating the vector field to compute a final set of positions. The difference between the visualization techniques described above is the order in which the integrations are performed. The streaklines take as input the current positions of all the particles, including those recently added at the seed points. All of the particles are 'moved' by integrating each one once using the data in the current time step. The particles may be rendered as individual points or connected in a way to simulate smoke. Particle paths take as input the seed point(s) and interatively integrate the particle position, incrementing the timestep with each integration. This results in an array of positions which is displayed as the particle path. Streamlines take as input the seed points and interatively integrate the particle position without incrementing the current timestep. This results in an array of positions which is displayed as the streamline.

In the case of streamlines and particle paths, it is the paths that are of interest, not the positions of individual points in the path. The researcher uses these tools to explore the flow field by moving the seedpoint and observing the path from that seedpoint. Thus the virtual environment system must be capable of computing the entire path in a single frame time. The usefulness of these tools are greatly enhanced when several paths can be computed within the required time.

Control over the seed points for all of the above tools are provided by lines of seed points called rakes. Rakes may be manipulated with the glove through finger gestures and hand motion. These rakes are grabbed at one of three points: center for rigid translation of the rake, or at either end for movement of that end of the rake. In this way rakes may be oriented in an arbitrary manner. Several rakes may be defined simultaneously The type and number of seedpoints in a particular rake is determined by the user. It has been found useful to use rakes of several different types in combination when studying a flow.

The integrations described above are made more complicated by the fact that the fluid flow data are provided on curvilinear grids, which contain the physical position of each grid point and the velocity vector at that point. If the position of a particle is known in physical space, a search of the curvilinear grid must be performed to locate the grid coordinates nearest that point and obtain the corresponding velocity vector data. This search involves unacceptable performance overhead. It is avoided in the virtual windtunnel by converting the velocity data to grid coordinates and performing all integrations in grid coordinates. The resulting paths are easily converted to physical coordinates by using their known grid coordinates to directly lookup their corresponding physical coordinates, using trilinear interpolation if necessary.

The computation of the visualization tools involves integrating particles throughout the flow field. Thus the data of the flowfield must be quickly accessible for the integration to be performed at the speeds required. Construction of particle paths in particular require the entire data set for all timesteps, as the particle paths may extend throughout the entire data set. In the stand-alone virtual windtunnel the flow data is placed in physical memory so that it can be randomly accessed with little time penalty. As the physical memory of our workstation contains 256 megabytes of memory, the data sets that can be examined with the stand-alone virtual windtunnel are limited to about 250 megabytes in size. As described in section 1, this is a severe constraint and prevents us from examining many interesting flows.

3: The Virtual Environment Interface

The virtual environment interface provides a natural three-dimensional environment for both display and control of rakes. This interface allows intuitive exploration of rich, complex geometries. It is very similar to the interface used in the stand-alone virtual windtunnel [1]. The basic components of the environment are a high-performance graphics workstation for computation and rendering, a BOOM for display, and a VPL Dataglove for control (figure 4).

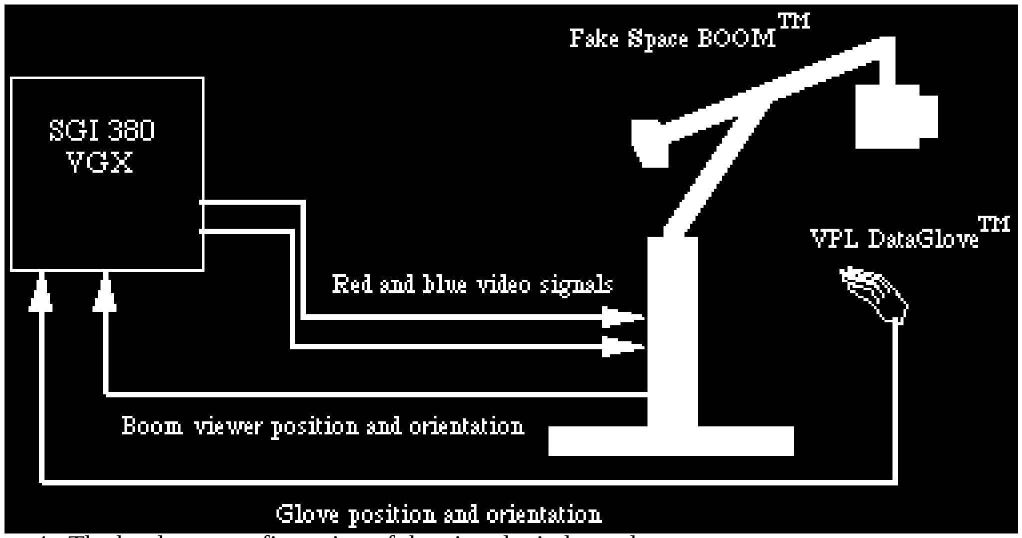
Fake Space BOOM™
SGI 380 VGX
VPL DataGlove™
Red and blue video signals
Boom viewer position and orientation
Glove position and orientation

Figure 4: The hardware configuration of the virtual windtunnel system.

The display for our virtual environment is the BOOM, manufactured by Fake Space Labs of Menlo Park CA., and fashioned after the prototype developed earlier by Sterling Software, Inc. at the VIEW lab at NASA Ames Research Center [6]. (see figure 5).

The boom is an alternative to the popular head-mounted LCD display systems that were pioneered at the VIEW lab [5] and are now widely used. The main advantage of the boom is that real CRTs can be used for display which provides much better brightness, contrast, and resolution than standard liquid crystal displays. Two monochromatic RS170 CRTs are provided, one for each eye, so that the computer generated scene may be viewed in stereo. The CRTs are viewed through wide field optics provided by LEEP Optics, so the computer generated image fills the user's field of view. The weight of the CRTs are borne by a counterweighted yoke assembly with six joints, which are designed to allow easy movement of the head with six degrees of freedom within a limited range. Optical encoders on the joints of the yoke assembly are continuously read by the host computer providing six angles of the joints of the yoke. These angles are converted into a standard 4x4 position and orientation matrix for the position and orientation of the BOOM head by six successive translations and rotations. By inverting this position and orientation matrix and concatenating it with the graphics transformation matrix stack, the computer generated scene is rendered from the user's point of view. As the user moves, that point of view changes in real-time, providing a strong illusion that the user is viewing an actual three-dimensional environment.

For user control in our virtual environment, the user's hand position, orientation, and finger joint angles are sensed using a VPL dataglove™ model II, which incorporates a Polhemus 3Space™ tracker. The Polhemus tracker gives the absolute position and orientation of the glove relative to a source by sensing multiplexed orthogonal electromagnetic fields. The degree of bend of knuckle and middle joints of the fingers and thumb of the user's hand are measured by the VPL Dataglove™ model II using specially treated optical fibers. These finger joint angles are combined and interpreted as gestures. The glove requires recalibration for each user, and the polhemus tracker has limited accuracy and is sensitive to the ambient electromagnetic environment. The dataglove works satisfactorily with user training and within a limited range.

The keyboard and mouse are also used as input devices to the virtual environment. The user can easily swing the boom away and interact with the computer in the usual way.

The local computation and rendering for our virtual environment is provided by the same Silicon Graphics Iris described in section 2.

Stereo display on the boom is handled by rendering the left eye image using only shades of pure red (of which 256 are available) and the right eye image using only shades of pure blue. When the blue (second, right-eye) image is drawn, it is drawn using a "writemask" that protects the bits of the red image. The Z-buffer bit planes are cleared between the drawing of the left- and right-eye images, but the color (red) bit planes are not cleared. Thus, the end result is separately Z-buffered left- and right-eye images, in red and blue respectively, on the screen at the same time with the appropriate mixture of red and blue where the images overlap.

The 1024x1280 pixel RGB video output of the VGX is converted into RS170 component video in real time using a scan converter. The red RS170 component is fed into the left eye of the boom, and the blue RS170 component into the right eye. The sync is fed to both eyes. Since the boom CRTs are monochrome, we see correctly matched (stereo) images.

4: Distributed Library

The main vehicle for distributing computation between supercomputers and workstations in the distributed virtual windtunnel is Distributed Library (dlib) [3]. The workstation acts as a client and the supercomputer acts as the server. Like many systems which provide for distributed processing, dlib is a high level interface to network services based on the remote procedure call (RPC) model [7][8][9][10]. However, unlike most of these systems, dlib was developed to provide a service which allows for a conversation of arbitrary length within a single context between client and server. The dlib server process is designed to be capable of storing state information which persists from call to call, as well as allocating memory for data storage and manipulation. While RPC protocols are frequently likened to local procedure calls without side effects, dlib more closely resembles the extension of the process environment to include the server process.

The use of dlib is much like developing a library of routines, say, an I/O library, on a local system. Application code is linked to routines in an I/O library. The I/O library contains simple routines which give access to the I/O devices controlled by the operating system device drivers (figure 6). The I/O device drivers in turn control the somewhat more complicated exchange of data with external devices.

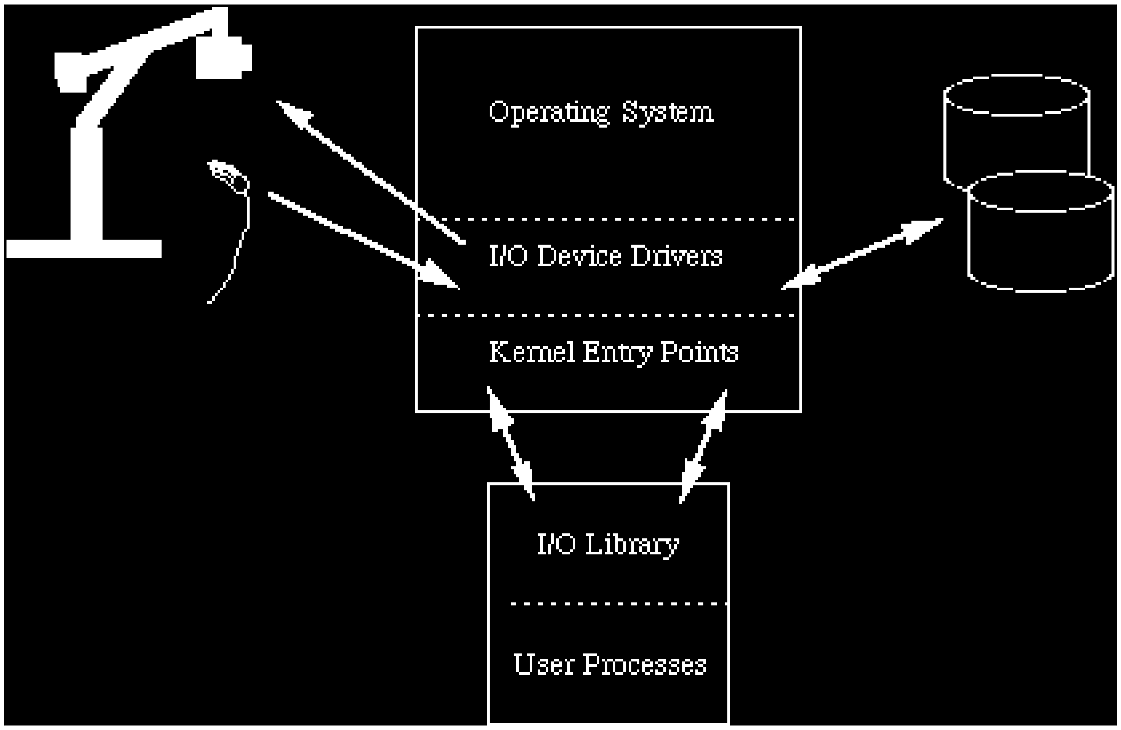

Figure 6: Access to local I/O devices using local I/O Library in, e. g., the stand-alone virtual wind-tunnel.

To execute a routine on a remote host, all the information necessary to execute the routine in the remote environment must be transmitted over the network to a remote server process. After execution of the routine is invoked, results of the execution must also be transmitted back to the local client process. Dlib provides utilities to automatically create the code which performs the network transactions required to invoke and execute the routine in the remote environment and exchange information between the client and server processes.

Due to the persistent nature of the remote environment, dlib is able to coordinate allocation and use of remote memory segments and provide access to remote system utilities. The application, through dlib, can "link" to the remote system's I/O library, for example, to utilize the remote system's I/O devices. (figure 7). The illustrated client process can utilize the monitor and mouse via the local I/O library and operating system. The client process can also utilize the remote disk via dlib which communicates to a remote server process. The remote server process has access to the remote disk via the remote I/O library.

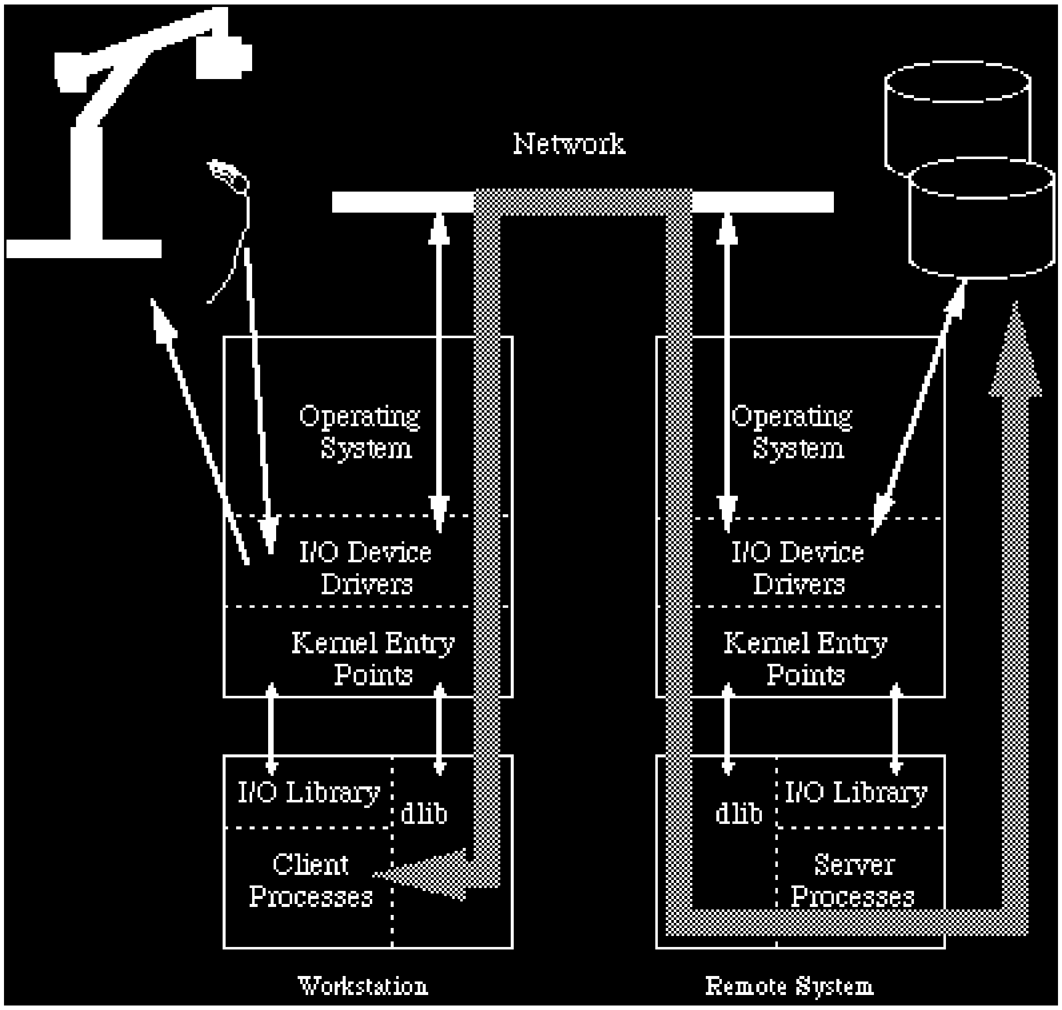

Figure 7: Access to remote I/O devices using dlib and remote I/O Library in the distributed virtual windtunnel. The solid lines represent actual data paths, while the stippled line is the effective data path from the client processes to the remote disk.

The distributed virtual windtunnel uses dlib to allow the workstation clients to gain access to the supercomputer's processing power, large memory, large disk storage capacity, and fast disk access.

Dlib was originally designed on a model of one client to one server. To allow multiple clients to share the server process environment, the dlib server was modified to accept more than one connection. Each connection is selected for service by the server process in the sequence that the dlib calls are received. The dlib calls are executed by the server in a single process environment as though there were only one client.

5: Implementation of the Distributed Virtual Windtunnel

5.1 Design Considerations

The performance constraints discussed in section 1 determine many aspects of a distributed virtual windtunnel implementation. In choosing a remote computer system, computational speed, accessibility, disk bandwidth, and compatibility with the stand-alone system must all be considered. Further design choices are motivated by available high-speed networks and the local workstations. What information is sent across networks is designed so that several workstations and virtual environment systems can access the same data on the host system. The format of the information is driven by the demand that as little data is transferred over the networks as possible. Finally the software architecture should be designed so that the individual computer systems perform the tasks that they are best suited for, and there are as few bottlenecks as possible.

The choice of the Convex C3240 system is motivated by several considerations. The primary consideration is the availability of this system, which is dedicated to visualization of large data sets in the NAS project at NASA Ames. While the NAS project also has two Cray supercomputers, they are heavily used, so interactive response time cannot be guaranteed. Dedicated time on the Convex C3240 has been made available for this project. The Convex C3240 has four vector processors. These processors can process vector arrays of up to 128 entries in length. The high performance of the Convex relies on the vectorization of code, which will be discussed in section 5.3. Our Convex has one gigabyte of physical memory and 100 gigabytes of disk storage. The disk bandwidth has been measured to be between 30 and 50 megabytes/second sustained rate, depending on the size of the file being read.

Two workstations are used for the local computation and rendering of the visualizations. This allows the implementation of a shared virtual windtunnel. The workstations are the Silicon Graphics (SGI) Iris 380GT VGX workstation described above and the Silicon Graphics Iris 440GT Skywriter. The skywriter has four 40 MHz MIPS R3000 CPUs with R3010 floating point chips and two independent VGXT graphics rendering pipelines. The Skywriter is used to render the virtual environment to the two-color channel BOOM IIC. These workstations have been chosen because they are available and used in stand-alone virtual windtunnel development. For our current implementation of the distributed virtual windtunnel, a workstation with two processors and a high-performance graphics pipeline would suffice.

The UltraNet high-speed network was chosen for the communication between the workstations and the remote system. This network is rated at 100 megabytes/second, but the UltraNet VME interface to the SGI workstation limits the bandwidth to 13 megabytes/second. This rate should be sufficient for most visualizations. As of this writing, the actual network performance is only 1 megabyte/second due to software bugs and the lack of a HIPPI interface for the Convex. Both of these problems are being addressed and we fully expect 13 megabytes/second soon.

The specification of the data that is returned by the remote system is determined by the requirements of minimal data transfer and that each computer do what it does best. The remote system computes the visualization tools as described in section 2, and sends the resulting paths out the network as arrays of floating point vectors in three dimensions. The workstation receives these arrays and renders them from the point of view determined by that workstation's virtual environment interface. This requires the transfer of 12 bytes per point in each array. Experience has shown that typical visualization scenarios involve tens of thousands of particles implying the transfer of several hundred thousand bytes of data per timestep. One might consider projecting these vectors to compute their screen coordinates on the remote system and sending this data to the workstation. This would reduce the transfer to eight bytes/point. In the virtual environment scenario, however, stereoscopic display requires two projections per point which implies 16 bytes/point. Thus sending the three-dimensional position of the points with 12 bytes/point in the arrays is optimal. The network transfer rates required for an update rate of ten frames/second neglecting processing and rendering overhead are summarized in table 1.

# of particles	# of bytes transferred	Required bandwidth for 10 fps (Mbytes/sec)
10,000	120,000	1.144
50,000	600,000	5.722
100,000	1,200,000	9.537

Table 1: Network constraints

In the specification of what information is sent to the remote system, the desire for a shared environment capability was the primary consideration. Ideally, at any time during the use of the distributed virtual windtunnel another workstation with a virtual environment interface should be able to "sign up" and interact with the already existing virtual environment. This requires that control over all objects in the virtual environment take place on the remote system. Thus the information that is sent to the remote system are those user commands which effect the virtual environment. These include hand position, hand gestures, keyboard and mouse commands, and any other control data that may be implemented. In the shared scenario, the position of the users' heads would also be sent so that they may be displayed as part of the virtual environment, indicating to participants in the environment where everyone is.

In this design, the information about the virtual control devices such as rakes must also be sent from the remote system to the workstation so that the current state of these devices may be correctly rendered. This adds a small overhead to the transfer of the visualization data described above, but this is typically minor compared to the visualization data itself.

Because the computation of the environment state is performed by a single machine, possible conflicting commands from different workstations are easily handled. As described in section 4, the dlib process on the remote system that handles the network traffic takes the input from various workstations in serial order. This allows conflicts to be resolved by a 'first come first served' rule. For example, if two users grab the same rake, the user who grabbed it first gets control of that rake and the second user is locked out of interaction with that rake until the first user lets the rake go. Other rakes are unaffected by this locking, so the second user can interact with them. Other control conflicts can be similarly resolved.

The problem of large data sets can be handled in a variety of ways. With one gigabyte of physical memory, data sets can be loaded into memory that are four times as large as in the stand-alone virtual windtunnel case. This allows for the visualization of interesting data sets not available to the stand-alone virtual windtunnel. Having the entire data set resident in memory is the easiest method of managing the data. When the data sets are larger than physical memory, however, the data must reside on a mass storage device, usually disk. The Convex C3240 with its disk I/O bandwidth of 30 megabytes/second can load datasets of up to about three and a quarter megabytes in 1/8th of a second. Thus datasets of whose timesteps are this size are limited only by the disk storage space. There are many interesting data sets which are still larger than this, however. An example is the hovering Harrier data set computed at NASA Ames, whose flow velocity data has about 36 megabytes per timestep. Visualization of this data set will require a disk bandwidth of about 600 megabytes per second. Thus we are still a long way from interactively visualizing very large unsteady data sets in a virtual environment. The disk bandwidth bottlenecks for an update rate of ten frames/sec are summarized for velocity grids in Figure 2:

# of points in grid	# of bytes in a timestep	# of timesteps that fit in a gigabyte	required disk bandwidth (Mbytes/sec)
131,072 (tapered cyl.)	1,572,864	682	15
436,906 (current max)	5,242,880	204	50
1,000,000	12,000,000	89	114.4
3,000,000	36,000,000	29	343.32
10,000,000	360,000,000	2 (actually 2.9)	3,433.2

Table 2: Disk bandwidth constraints

Loading timesteps from disk as described in the previous paragraph precludes the computation of particle paths of arbitrary length in real time, as they require a different timestep for every point in the path. All the timesteps required for the computation of a particle path must be resident in memory. Thus the number of timesteps that can fit in physical memory places a limit on the length of the particles paths. The timestep that would be loaded into memory in this case would be the current timestep plus the maximum particle path length.

A final consideration is the desire that the development of the stand-alone and distributed virtual windtunnels be as close as possible. In this way development of the visualization tools and environment can occur on both systems in parallel. In the current implementation both the distributed and stand-alone versions are compiled from the same source code using DEFINE statements where differences in architecture occur. This is possible because of the compatibility of the Convex C3240 and SGI internal achitectures, specifically the sharing of the IEEE floating point number format (a compile time option on the Convex).

5.2 System Architecture

In the design motivated by the above considerations, each workstation reads its input devices and sends their commands to the remote system. The remote system updates the virtual environment including if necessary loading the data for the current timestep, computes the current visualizations, and transfers the environment state back to the workstations. Each workstation renders this state to its virtual environment display device. Many of these tasks can be performed in parallel.

On the remote system, computation of the visualizations can occur while the data from the previous computation is sent to the network (figure 8). If the timesteps are being loaded from disk, that loading can also occur in parallel. The timestep required for the next computation is loaded into a buffer. This parallelization will impact the time required for the computation so a careful study must be performed to determine the optimal balance of tasks.

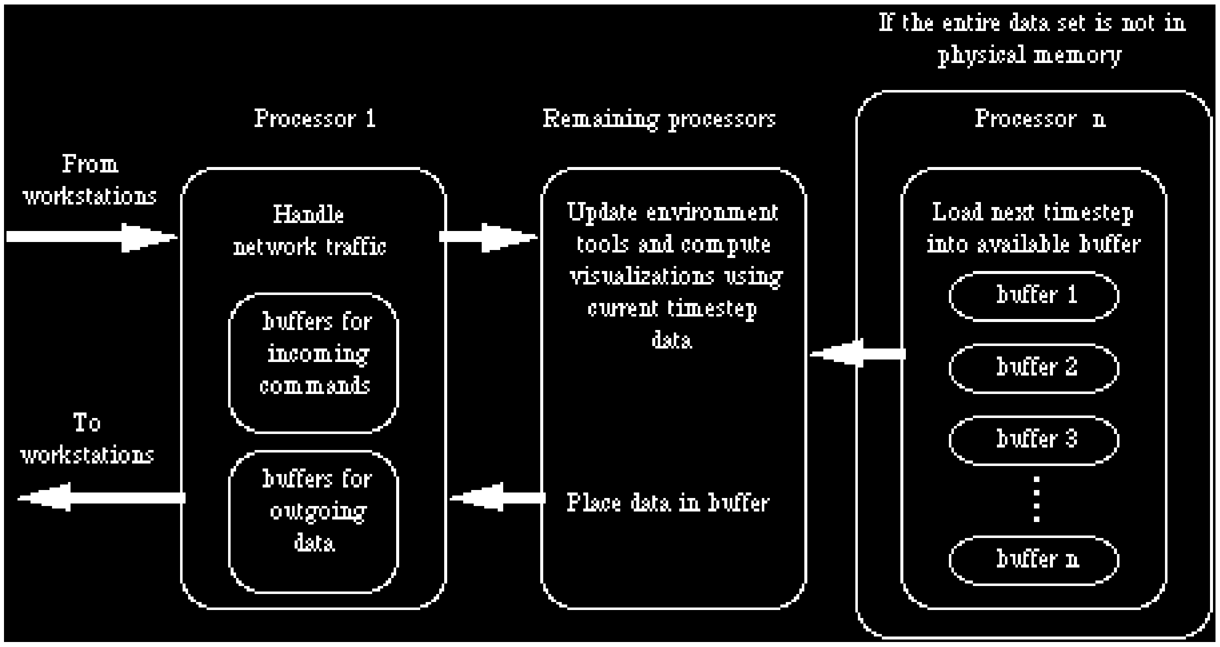
If the entire data set is not in physical memory
Processor 1
Remaining processors
Processor n
From workstations
Handle network traffic
buffers for incoming commands
To workstations
buffers for outgoing data
Update environment tools and compute visualizations using current timestep data
Place data in buffer
Load next timestep into available buffer
buffer 1
buffer 2
buffer 3
buffer n

Figure 8: The software architecture of the remote system. The data flow on the left side of the figure is over the Ultra network. The interprocess communication is via shared memory. The rightmost process is present only when the data set is not stored in physical memory.

On the workstation, at least two processors are desirable so the rendering of the graphics and the handling of the network traffic can be run in parallel (figure 9). In this way the graphics performance is not tied to the network and remote computation performance, so the head-tracked display of the virtual environment can run at very high rates. Note that even though the head-tracked display is updating at very high rates, the entire computation cycle is still required to take place in less than 1/8th of a second due to the user interaction with the visualization tools.

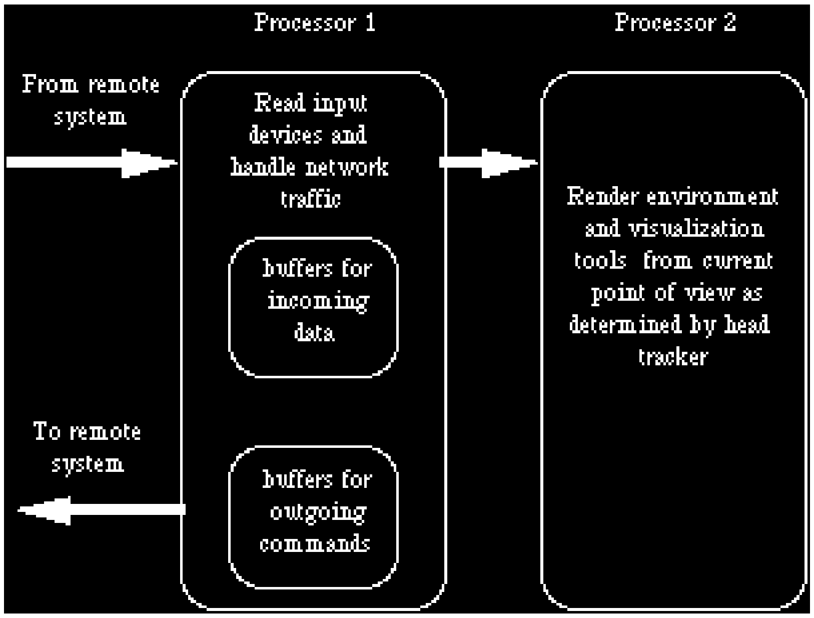
Processor 1
Processor 2
From remote system
Read input devices and handle network traffic
buffers for incoming data
Render environment and visualization tools from current point of view as determined by head tracker
To remote system
buffers for outgoing commands

Figure 9: The software architecture of the workstation. The data flow on the left side of the figure is over the Ultra network. The interprocess communication is via shared memory.

5.3 Optimization

To take advantage of the computational power of the Convex, considerable code optimization is required. The main body of the computational code must be vectorized so that the full power of the vector registers can be used. The integration algorithm for the computation is second-order Runge-Kutta, which requires two accesses of the vector field data from memory each involving eight floating point loads to set up for trilinear interpolation, two trilinear interpolations, and two simple computations per component per point integrated. To convert the current point's location from grid coordinates to physical coordinates, another 8 floating point loads plus a trilinear interpolation are required per component per point.

A conflict arises between optimization for fast non-vectorized (scalar) execution and optimization for vectorized execution. Our system is developed in C. The use of pointers and striding techniques which produce optimized scalar C code prevents the vectorization of that code. The use of standard C arrays allows vectorization but the code is somewhat less optimal. When vectorization actually occurs, the tradeoff is clearly in favor of vectorization.

To evaluate the computational performance, a benchmark computation of 100 streamlines each containing 200 points was performed. This scenario contains 20,000 points with a transfer over the networks of 240,000 bytes of data. The code as originally written for the stand-alone virtual windtunnel uses optimized scalar C techniques such as pointer manipulation and striding. This code successfully parallelizes across the four processors of the Convex 440 by distributing the streamlines among the processors. In this case the benchmark computation can be performed in about 0.24 seconds. This is slower than the performance of the stand-alone virtual windtunnel distributing the streamlines across 8 processors, which performed the same computation in 0.13 to 0.14 seconds.

An attempt was made to vectorize the computation over the streamlines. This is the only possibility, as the computation of an individual streamline is an iterative process. Each component of each point in the streamline is handled in parallel by different processors. Thus three processors are used in this vectorized computation. Vectorized in this way the benchmark computation can be performed in about 0.19 seconds. This lack of performance can be understood in terms of the many memory accesses that are performed in each computation as discussed above. This in combination with the effective loss of one processor explains the poor performance of the vectorized code compared to the non-vectorized but parallelized code.

The further optimization of the computation is under study. One optimization is to parallelize across groups of streamlines and vectorize across streamlines in a group. Generally, the speed of the computation places a limit on particle number. Examples of this constraint for various performance parameters are shown in table 3 for a frame rate of ten frames/second, assuming that the performance scales with the number of particles:

Benchmark performance	maximum # of particles	# of streamlines w/ 200 particles
0.25 seconds	8,000	40
0.19 seconds (current)	10,526	52
0.13 seconds (workstation)	15,384	76
0.10 seconds	20,000	100
0.05 seconds	40,000	200

Table 3: Computational performance constraints.

Comparison with the previous table shows that once the UltraNet is performing as expected, computation time on the remote machine will be the primary constraint on the number of particles that can be used for visualization.

6: Conclusions

This paper describes an initial implementation of an interactive distributed virtual environment. This implementation has been successful in several ways. The feasibility of real-time virtual environment interaction over a high-speed network involving transfers of hundreds of kilobytes of data has been demonstrated. Using the large memory of remote supercomputers, larger data sets have been visualized in a virtual environment than has been previously possible. An architecture which supports data sets read dynamically from disk and multiple virtual environment users has been designed. These successes suggest that distributed computation should be persued for the visualization of very large unsteady data sets. It is possible that with more work the computational power of the Convex C3240 will provide much higher performance virtual environments.

The most significant constraint on the number of particles that can be used for visualization is the computation time. Replacing the Convex C3240 with a higher performance supercomputer such as a Cray Research Inc. system will perhaps relax this constraint.

Further work includes the extension of the computational algorithms to handle multiple grid data sets, optimization of the disk access for data sets that are stored on disk, optimization of the shared environment interactions, and development of greater user control over the virtual environment. Finally, the usefulness of virtual environments in the visualization of fluid flow must be formally studied.

While the Convex cannot provide the disk bandwidth required for very large data sets, the methods developed in this project will extend to those machines in the future which can provide the required performance. These methods are useful in contexts other than virtual environments, such as the visualization of unsteady flows in the conventional screen and mouse environment.

6: Acknowledgements

Much thanks and credit goes to Creon Levit for fruitful design discussions and planting many of the seeds that lead to this paper. Thanks also to Tom Woodrow and David Lane for assisting with the distribution to the Convex. Tom Lasinski is appreciated for his leadership and support of this project. Thanks to Jeff Hultquist and Tom Lasinski for helpful suggestions on early versions of this paper. Finally, thanks to the NAS division at NASA Ames for general support of this work.

References

[4] S. Bryson and C. Levit, "The Virtual Windtunnel: An Environment for the Exploration of Three-Dimensional Unsteady Fluid Flows", *Proceedings of IEEE Visualization '91*, San Diego, Ca. 1991, to appear in *Computer Graphics and Applications* July 1992

[2] D. Jespersen and C. Levit, “Numerical Simulation of Flow Past a Tapered Cylinder”, paper AIAA-91-0751, American Institute of Aeronautics 29th Annual Aerospace Sciences Meeting, Reno (1991).

[3] Yamasaki, M. Distributed library. NAS Applied Research Technical Report RNR-90-008 (Apr. 1990), NASA Ames Research Center.

[4] W.J. Yang (editor), *Handbook of Flow Visualization*, Hemisphere Pub., New York (1989)

[5] Fisher, S. et. al., Virtual Environment Interface Workstations, *Proceedings of the Human Factors Society 32nd Annual Meeting*, Anaheim, Ca. 1988

[6] I.E. McDowall, M. Bolas, S. Pieper, S.S. Fisher and J. Humphries, “Implementation and Integration of a Couterbalanced CRT-Based Stereoscopic Display for Interactive Viewpoint Control in Virtual Environment Application”, in *Proc. SPIE Conf. on Stereoscopic Displays and Applications*, J. Merrit and Scott Fisher, eds. (1990)

[7] Birrell, A. D., and Nelson, B. J. Implementing remote procedure calls. ACM Trans. on Comp. Sys. 2, 1 (Jan. 1984), 39-59.

[8] Dineen, T. H., Leach, P. J., Mishkin, N. W., Pato, J. N., and Wyatt, G. L. The network computing architecture and system: an environment for developing distributed applications. Proceedings of Summer Usenix (June 1987), 385-398.

[9] Sun Microsystems. Request for Comment #1057 Network Working Group (June, 1988).

[10] Xerox Corporation. Courier: the remote procedure call protocol. Xerox System Integration Standard (XSIS) 038112, (Dec. 1981).

Figure 1: Streaklines of the flow around the tapered cylinder rendered as smoke.

Figure 2:   Streamlines of the flow around the tapered cylinder.

Figure 3: Streamlines of the flow around the tapered cylinder from the same seedpoints as in figure 2, but at a later time.

Figure 5: Boom and glove hardware interface to the Virtual Windtunnel